Contents

Installing the Sidecar

6 minute read page test

Injection

In order to take advantage of all of Istio's features, pods in the mesh must be running an Istio sidecar proxy.

The following sections describe two ways of injecting the Istio sidecar into a pod:

enabling automatic Istio sidecar injection in the pod’s namespace, or by manually using the `istioctl` command.

When enabled in a pod’s namespace, automatic injection injects the proxy configuration at pod creation time using an admission controller.

Manual injection directly modifies configuration, like deployments, by adding the proxy configuration into it.

If you are not sure which one to use, automatic injection is recommended.

Automatic sidecar injection

Sidecars can be automatically added to applicable Kubernetes pods using a mutating webhook admission controller provided by Istio.

> While admission controllers are enabled by default, some Kubernetes distributions may disable them. If this is the case, follow the instructions to turn on admission controllers.

When you set the `istio-injection=enabled` label on a namespace and the injection webhook is enabled, any new pods that are created in that namespace will automatically have a sidecar added to them.

Note that unlike manual injection,

automatic injection occurs at the pod-level. You won't see any change to the deployment itself. Instead, you'll want to check individual pods (via `kubectl describe`) to see the injected proxy.

Deploying an app

Deploy sleep app. Verify both deployment and pod have a single container.

```
$ kubectl apply -f @samples/sleep/sleep.yaml@
$ kubectl get deployment -o wide
NAME    READY   UP-TO-DATE   AVAILABLE   AGE   CONT
AINERS   IMAGES                  SELECTOR
sleep   1/1     1            1           12s   slee
p        curlimages/curl         app=sleep
```

```
$ kubectl get pod
NAME                     READY   STATUS    RESTARTS
  AGE
sleep-8f795f47d-hdcgs   1/1     Running   0
  42s
```

Label the `default` namespace with `istio-injection=enabled`

```
$ kubectl label namespace default istio-injection=enabled --overwrite
$ kubectl get namespace -L istio-injection
NAME                 STATUS   AGE     ISTIO-INJECTION
default              Active   5m9s    enabled
...
```

Injection occurs at pod creation time. Kill the running pod and verify a new pod is created with the injected sidecar. The original pod has `1/1 READY` containers, and the pod with injected sidecar has `2/2 READY` containers.

```
$ kubectl delete pod -l app=sleep
$ kubectl get pod -l app=sleep
pod "sleep-776b7bcdcd-7hpnk" deleted
NAME                     READY     STATUS        RE
STARTS   AGE
sleep-776b7bcdcd-7hpnk   1/1       Terminating   0
         1m
sleep-776b7bcdcd-bhn9m   2/2       Running       0
         7s
```

View detailed state of the injected pod. You should see the injected `istio-proxy` container and corresponding volumes.

```
$ kubectl describe pod -l app=sleep
...
Events:
  Type    Reason     Age   From               Messa
ge
  ----    ------     ----  ----               -----
--
  ...
  Normal  Created    11s   kubelet            Creat
ed container istio-init
  Normal  Started    11s   kubelet            Start
ed container istio-init
  ...
  Normal  Created    10s   kubelet            Creat
ed container sleep
  Normal  Started    10s   kubelet            Start
ed container sleep
  ...
  Normal  Created    9s    kubelet            Creat
ed container istio-proxy
  Normal  Started    8s    kubelet            Start
ed container istio-proxy
```

Disable injection for the `default` namespace and verify new pods are created without the sidecar.

```
$ kubectl label namespace default istio-injection-
$ kubectl delete pod -l app=sleep
$ kubectl get pod
namespace/default labeled
pod "sleep-776b7bcdcd-bhn9m" deleted
NAME                     READY     STATUS        RE
STARTS   AGE
sleep-776b7bcdcd-bhn9m   2/2       Terminating   0
         2m
sleep-776b7bcdcd-gmvnr   1/1       Running       0
         2s
```

Controlling the injection policy

In the above examples, you enabled and disabled injection at the namespace level. Injection can also be controlled on a per-pod basis, by configuring the `sidecar.istio.io/inject` label on a pod:

Resource	Label	Ena

Namespace	`istio-injection`	`enab`
Pod	`sidecar.istio.io/inject`	`"tru`

If you are using control plane revisions, revision specific labels are instead used by a matching `istio.io/rev` label. For example, for a revision named `canary`:

Resource	Enabled label	
Namespace	`istio.io/rev=canary`	`istio-in`
Pod	`istio.io/rev=canary`	`sidecar.`

If the `istio-injection` label and the `istio.io/rev` label are both present on the same namespace, the `istio-injection` label will take precedence.

The injector is configured with the following logic:

1. If either label is disabled, the pod is not injected
2. If either label is enabled, the pod is injected
3. If neither label is set, the pod is injected if `.values.sidecarInjectorWebhook.enableNamespacesByDefault` is enabled. This is not enabled by default, so generally this means the pod is not injected.

Manual sidecar injection

To manually inject a deployment, use `istioctl kube-inject`:

```
$ istioctl kube-inject -f @samples/sleep/sleep.yaml
@ | kubectl apply -f -
serviceaccount/sleep created
service/sleep created
deployment.apps/sleep created
```

By default, this will use the in-cluster configuration. Alternatively, injection can be done using local copies of the configuration.

```
$ kubectl -n istio-system get configmap istio-sidec
ar-injector -o=jsonpath='{.data.config}' > inject-c
onfig.yaml
$ kubectl -n istio-system get configmap istio-sidec
ar-injector -o=jsonpath='{.data.values}' > inject-v
alues.yaml
$ kubectl -n istio-system get configmap istio -o=js
onpath='{.data.mesh}' > mesh-config.yaml
```

Run `kube-inject` over the input file and deploy.

```
$ istioctl kube-inject \
    --injectConfigFile inject-config.yaml \
    --meshConfigFile mesh-config.yaml \
    --valuesFile inject-values.yaml \
    --filename @samples/sleep/sleep.yaml@ \
    | kubectl apply -f -
serviceaccount/sleep created
service/sleep created
deployment.apps/sleep created
```

Verify that the sidecar has been injected into the sleep pod with `2/2` under the READY column.

```
$ kubectl get pod  -l app=sleep
NAME                     READY   STATUS    RESTARTS
   AGE
sleep-64c6f57bc8-f5n4x   2/2     Running   0
   24s
```

Customizing injection

Generally, pod are injected based on the sidecar injection template, configured in the `istio-sidecar-injector` configmap. Per-pod configuration is available to override these options on individual pods. This is done by adding an `istio-proxy` container to your pod. The sidecar injection will treat any configuration defined here as an override to the default injection template.

Care should be taken when customizing these settings, as this allows complete customization of the resulting `Pod`, including making changes that cause the sidecar container to not function properly.

For example, the following configuration customizes a variety of settings, including lowering the CPU requests, adding a volume mount, and adding a `preStop` hook:

```
apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
  containers:
  - name: hello
    image: alpine
  - name: istio-proxy
    image: auto
    resources:
      requests:
        cpu: "100m"
    volumeMounts:
    - mountPath: /etc/certs
      name: certs
    lifecycle:
      preStop:
        exec:
          command: ["sleep", "10"]
  volumes:
  - name: certs
    secret:
      secretName: istio-certs
```

In general, any field in a pod can be set. However, care must be taken for certain fields:

- Kubernetes requires the `image` field to be set before the injection has run. While you can set a specific image to override the default one, it is recommended to set the `image` to `auto` which will cause the sidecar injector to automatically select the image to use.
- Some fields in `Pod` are dependent on related settings. For example, CPU request must be less than CPU limit. If both fields are not configured together, the pod may fail to start.

Additionally, certain fields are configurable by annotations on the pod, although it is recommended to use the above approach to customizing settings.

Custom templates

(experimental)

> This feature is experimental and subject to change, or removal, at any time.

Completely custom templates can also be defined at installation time. For example, to define a custom template that injects the `GREETING` environment variable into the `istio-proxy` container:

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
metadata:
  name: istio
spec:
  values:
    sidecarInjectorWebhook:
      templates:
        custom: |
          spec:
            containers:
            - name: istio-proxy
              env:
              - name: GREETING
                value: hello-world
```

Pods will, by default, use the `sidecar` injection template, which is automatically created. This can be overridden by the `inject.istio.io/templates` annotation. For example, to apply the default template and our customization, you can set `inject.istio.io/templates=sidecar,custom`.

In addition to the `sidecar`, a `gateway` template is provided by default to support proxy

injection into Gateway deployments.